Design of a Fault-Tolerant Three-Dimensional Dynamic Random-Access Memory with On-Chip Error-Correcting Circuit

Pinaki Mazumder, *Member, IEEE*

Abstract— **As VLSI technology is inching forward to the ultimate limits of physical dimensions, memory manufacturers are striving to integrate more memory cells in a chip by employing innovative three-dimensional cell topographies. Most of the current-generation multimegabit dynamic random-access memory (DRAM) chips use three-dimensional storage capacitors where the charge is stored on a vertically integrated trench-type structure. It has been experimentally verified that these memory cells have poor reliability, because they are highly vulnerable to alpha particles, which frequently create plasma shorts between two adjoining trench capacitors on the same word line, resulting in uncorrectable double-bit soft errors. The conventional on-chip error-correcting codes (ECCs) cannot correct such double-bit/word-line soft errors. The paper presents a systematic study of soft-error related problems and it discusses the methodologies to correct single-bit and double-bit memory-cell upsets by using on-chip ECC circuits. Conventional double-error-correcting (DEC) codes used in digital communications are known to be inadequate for this application. By modifying the product code, an effective coding scheme has been designed in this paper that can be integrated within a DRAM chip to correct double-bit errors. The paper demonstrates that the reliability of a memory chip can be improved by several million times by integrating the proposed circuit. The area and timing overhead have been calculated and compared with the memory chips without any ECC and chips with single-error-correcting (SEC) codes. The ability of the circuit to correct soft errors in the presence of multiple-bit errors has also been analyzed by combinatorial enumeration.**

Index Terms— **Augmented product code, dynamic random-access memory (DRAM), error-correcting code, projective geometry code, trench capacitor.**

I. Introduction

As the feature width of very-large-scale-integration (VLSI) technology is rapidly approaching its physical limits of about 0.35 μm [1], the dynamic random-access memory (DRAM) size is quadrupling every three years or so to grow to its estimated size of 4 Gb [2]. Already many DRAM manufacturers [3], [6] have experimented with 64-Mb three-dimensional DRAM's with vertically-mounted, trench-type storage capacitors. This enormous propect of gargantuan, high-density memory has posed two formidable challenges to the memory designers: a) how to test memory chips economically during fabrication, and b) how to ensure high reliability in storage data during the operational life cycle of a memory chip. A number of researchers [7]–[9] have addressed the first issue by proposing efficient design-for-testability DRAM architectures where a group of cells are tested in a single memory cycle. The main objective of this paper is to address the second issue, namely how to concurrently detect and correct the soft errors by employing an on-chip error-correcting circuit. The soft errors are predominantly caused by alpha particles, and sometimes they result from transients like power-supply voltage spikes, thermal effects, and man-made static. These errors are called soft because they do not damage the physical function of a cell permanently, and they can be easily corrected by inverting the data in the faulty cells. In contrast, the errors that result from common functional faults such as stuck-at, coupling, and pattern-sensitive are classified as hard and medium. The permanent faults, e.g., stuck-at, result in hard errors, while the coupling faults and the pattern-sensitive faults usually result in medium errors, because they are difficult to correct [10]. Comprehensive parallel test algorithms proposed in [11] can efficiently detect all hard and medium errors in high-density DRAM.

It is well known that if alpha particles are incident on the intervening space between two adjoining trench capacitors, the resulting plasma discharge may delete data in both the capacitors. Chern *et al.* [12] have done extensive Monte Carlo simulation to study the charge-sharing mechanism caused by an alpha-particle-induced plasma short between capacitors. Their study conclusively proved that trench capacitors are likely to cause double-bit upsets. These double-bit errors cannot be corrected by the conventional single-error-correcting (SEC) coding circuits. A novel fault-tolerant DRAM design is, therefore, proposed to correct double-bit errors on every word line within the chip. Thus, in an n-bit DRAM organized into s square subarrays each of size $\sqrt{n/s} \times \sqrt{n/s}$, the proposed design can correct as many as $2\sqrt{sn}$ errors. The improvement in soft-error rate (SER) as a result of the proposed code is found to be better than 10^6, and thereby the reliability of the memory improves considerably.

Manuscript received April 8, 1988; revised January 10, 1989 and January 13, 1993. This work was supported in part by the National Science Foundation under Grant MIPS 9013092 and by the URI Program of the Army Research Office under Grant DAAL 03-87-K-0007.

The author is with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109-2122.

IEEE Log Number 9212698.

0018–9340/93$03.00 © 1993 IEEE

II. Characterization of Alpha-Particle-Induced Soft Errors

In a DRAM chip more than 98% of failures that occur during the normal operation are radiation-induced soft errors [13]–[15]. On-line hard and medium errors are relatively very low in a well-designed chip. A small fraction of failures come from transients like voltage spikes and man-made statics. By introducing suitable filtering circuits these sources of transient errors can be sufficiently suppressed. But the alpha-particle-induced soft errors are becoming more and more critical as the cell dimension is reducing. The capacitor value and the cell topography play an important role in deciding the soft-error rate (SER). In this section, a detailed study is made to identify the sources of alpha particles, their effects on three-dimensional DRAM with trench-type capacitors, and how to minimize these effects by using appropriate shielding mechanisms.

There are three radiation sources for causing soft errors in a DRAM chip. The cosmic rays in the atmosphere may strike the chip with sufficient impinging velocity to generate electron-hole pairs that may contribute to the soft errors. For space and avionics applications, this cosmic radiation is a major concern, and suitable radiation-hardened protective measures are adopted to minimize these effects. The second source of alpha particles is the radioactive decay of uranium and thorium, contained in minute proportions within the packaging material [16], [17]. This is a relatively large flux, but it can also be reduced sufficiently by coating the chip with radiation-hardened film that shields the chip from packaging material. The third source of the alpha particles is the radioactive impurities in the materials within the chip itself. Small traces of thorium and uranium are found in the metal and in the silicon substrate. The alpha-particle emission from these residual radioactive impurities frequently cause soft errors in a memory chip. Unlike the first two sources of alpha particles, the soft errors from residual alpha-activity cannot be effectively controlled by using protective films. Several studies are being made on the cell topography [18] and bit-line design [19] that will improve SER. But so far no effective way has been found to eliminate this residual alpha activity.

The memory plane of a DRAM chip is the most sensitive to upset the alpha particle hits. The peripheral logic and decoder are usually robust to alpha particles, and very rarely contribute to soft errors. Moreover, such errors can be easily corrected by data retry since the transient faults in such circuits are usually combinational. The two most sensitive structures in a memory plane are the bit lines and the storage cells. The failure mechanisms in these structures are conceptually different, and they will be called here the *memory-cell mode upset* and the *bit-line mode upset.* In Section II-B, it is pointed out that the memory-cell mode upset caused by noise electrons flowing into one or more storage capacitors usually result in sequential error, which cannot be corrected by data retry. Suitable error-correcting coding circuits are designed in this paper to eliminate these soft errors. The bit-line mode upset is caused by electron-hole pairs impinging on a floating bit-line in a read cycle. Such failure mode usually results in a combinational fault where a read operation is faulty because of incorrect charge level on a bit line. The SER from the bit-line failure mode is inversely proportional to read cycle time, but the SER due to the memory-cell mode failure is independent of memory cycle time.

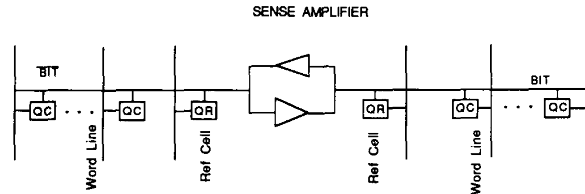


Fig. 1. Split array DRAM organization.

A. Bit-Line Mode Soft Error

A functional knowledge of the memory cycle is needed to understand the bit-line mode soft error. A typical organization of a DRAM chip utilizes the differential amplifiers for sensing the partitioning of each array into two identical subarrays as shown in Fig. 1. The bit line B_i is split into right-half B_i^R and left-half B_i^L. Data are stored in capacitors at all the crosspoints of different bit lines and word lines. A memory cell typically consists of a storage capacitor in series with an access transistor, which is selected by the word line connected to its gate. The content of a memory cell can be read by sensing its stored charge through the bit line and the differential sense amplifier. A read cycle consists of two distinct phases—the precharge phase and the sense phase. During the precharge phase, both left and right halves of bit lines are charged to a predetermined level, and after precharging is over both the bit lines remain in floating state. During the sensing phase, the charge sharing occurs between the selected cell and the bit line, and the sense amplifier differentiates the voltage level between two bit lines to determine whether the selected cell contains a 0 or 1. In Fig. 1, in the precharge phase, both the bit-line halves B_i^R and B_i^L are charged to a predetermined value V_P, which may be typically near to half the supply voltage. Each half of the bit line contain a reference cell having a fixed charge Q_R. The cell is said to contain a 1 if its capacitor stores a charge $Q_C \geq Q_R$, and it is said to contain a 0 if $Q_C < Q_R$. When a cell on $B_i^R\,(B_i^L)$ is selected for a READ operation, the word line $W^L\,(W^R)$ is activated simultaneously to select the reference cell on $B_i^L\,(B_i^R)$. In the sense phase, when the reference cell is selected, charge sharing occurs between $B_i^L\,(B_i^R)$ and the reference cell, and the new $B_i^L\,(B_i^R)$ voltage is $V^* \approx V_P$. Similarly, in the right-half (left-half) bit line, charge sharing occurs between the selected cell and $B_i^R\,(B_i^L)$, which consequently assumes a new voltage $V^R\,(V^L)$. The differential amplifier senses the voltage difference $\Delta V = V^R - V^*\,(V^L - V^*)$ to read the value of the selected cell. If ΔV is positive (negative) and is greater than the differential threshold voltage, v_{th}, then the selected cell is recognized to have 1 (0). This is illustrated in Fig. 2(a). If ΔV is smaller than v_{th}, the selected cell may be read incorrectly by the sense amplifier.

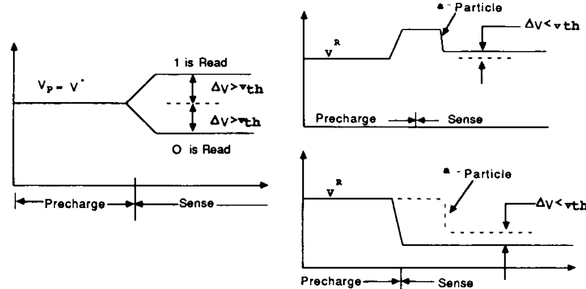


Fig. 2. Bit-line mode soft error.

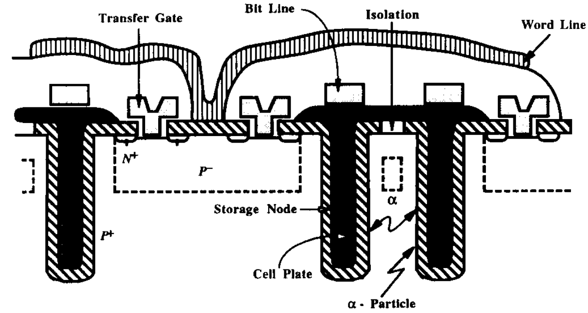


Fig. 3. Double-bit soft errors in three-dimensional DRAM cells.

The bit line is vulnerable to soft errors during the sensing phase, particularly in the interval from the start of the the sensing phase to sense amplifier latch-up when the bit line remains floating. During the sensing phase if an alpha particle strikes the bit line containing the reference cell, its charge may reduce, resulting in a new reference voltage $V^{**} \ll V^*$ as shown in Fig. 2(c). If the selected cell contains 0, and ΔV may become lesser than v_{th} resulting in a faulty read operation. On the other hand, during the sensing phase, if the alpha particle strikes the bit line containing the selected cell which contains 1, the bit-line voltage may degrade such that $\Delta V < v_{th}$. The resulting read operation may be erroneous, as shown in Fig. 2(b).

B. Memory-Cell Mode Soft Error

The cell topography plays an important role in deciding how many cells can fail even by the incidence of a single alpha particle. In planar implementation of the storage capacitor, the track of alpha particles usually restricts within the boundary of a single cell, and the resulting fault causes a single-event upset. Toyabe *et al.* [19], solved three-dimensional diffusion equations for alpha-particle-induced electrons to analyze the SER for memory-cell mode upset. This SER is directly proportional to the effective area (σ^2) of the memory cell and the incident alpha flux density (ϕ).

As the memory size is quadrupling, the cell dimension and the storage capacitor value are reducing by a factor of 2 [20]. The present three-dimensional DRAM employs a deep trench capacitor that extends from the planarized surface through the n-well into the p^+ substrate, as shown in Fig. 3. The capacitance is typically 30 fF, and the capacitor is highly susceptible to being discharged by noise electrons. The memory-cell mode soft errors can be principally classified into two types: single-cell upsets and double-cell upsets. In a single-cell upset, an alpha particle strikes on a single capacitor, discharging it alone. On the other hand, if the alpha particle strikes on the intervening space between two adjoining capacitors, a plasma discharge may occur between both the capacitors, resulting in alteration in charge level in both the capacitors. Such soft errors are known as double-cell upsets, and are very common in a three-dimensional DRAM chip. If σ^2 is the effective area of a cell and δ is the intercellular distance, then the probability of an alpha particle striking on the intervening space is proportional to δ/σ. Thus a large number of alpha particles may cause double-bit errors if they have sufficient kinetic energy to discharge both the capacitors. The Monte Carlo simulation done by Sai-Halasz *et al.* [15] has revealed that, as the feature width and the critical charge in storage capacitor decrease, the double-bit soft errors dominate over the single-bit errors. It may be emphasized that the earlier soft-error analysis was based on a single-event upset where the storage capacitors were planarized. The conventional on-chip and system-level error-correcting circuits are, therefore, SEC/DED type, which fails to correct the above double-cell upsets.

The objective of this paper is to design a new double-error correcting (DEC) coding circuit that can be integrated in a DRAM chip to reduce the SER and to improve the reliability of the memory system. It may be noted that in order for an error-correcting circuit (ECC) to be used for soft-error correction in DRAM cells with trench-type capacitors, it should satisfy the following requirements:

1) It should correct both the single-bit and the double-bit soft errors.
2) It should match the pitch-width constraints of closely juxtaposed bit-lines and word lines in the memory planes of DRAM chip.
3) It should be compatible with the organization of the multimegabit DRAM chip (which is usually organized into 8 to 16 subarrays [4]).
4) It should have low overhead—bit redundancy, layout complexity, and double-bit correction circuit.
5) Encoding and decoding circuit for the code should not be very complex.
6) It should not considerably deteriorate the memory access time.

III. A Double-Bit Error-Correcting and/or Detecting Code

The conventional error-correcting techniques employ system-level Hamming code, which can detect two errors and correct one error in a W-bit word (usually $W = 32$) by using $\log_2 W + 2$ redundant bits. Usually in a hierarchical memory system, with $N = kn$ words each of W-bit width, at most $k(W + \log_2 W + 2)$ DRAM chips, each of size n memory cells, are used to store NW information bits and $N(\log_2 W + 2)$ parity check bits. Thus every time a word in the memory system is accessed, only one cell in a

DRAM chip is sensitized. If the accessed cell is faulty, it is corrected by the error-correcting circuit. This technique has an acceptable uncorrectable-error rate (UER) for a small DRAM chip ($n \leq$ 16K). For a multimega-bit DRAM with $n \geq$ 4M, the fault latency[1] becomes very large and the UER increases rapidly to reduce the reliability of the memory system. To improve the UER, a number of cells within a DRAM chip are sensitized when a single bit is accessed and the faulty cells are corrected immediately. A number of on-chip error-correcting techniques have been proposed in DRAM literature. Osman [21] organized the cells in a DRAM chip into a number of blocks, and compared the block parities to correct a single-bit soft error. Mazumder and Patel [22] used a similar technique over a two-level memory system, and showed that the soft-error rate improved by a factor of 10^6. They used a parallel-signature analyzer (PSA) to test the chip in parallel, and reconfigured the PSA to detect the soft error. Nippon Telephone and Telegraph [3] used product code to correct a single-bit error in their experimental 16-Mb DRAM chip.

The main limitation of the product and Hamming codes is that they fail to correct double-bit errors, and in a three-dimensional DRAM where the double-bit soft errors are relatively common, these codes are inadequate for on-chip ECC applications. The conventional double-bit error-correcting codes such as Bose–Chaudhury–Hocquenghem (BCH) [23], Reed–Solomon [24], and Golay [23], cannot be readily applied to correct double-bit errors in a DRAM chip. These codes are frequently used in digital communications to correct t-bit ($t \geq 1$) errors. The encoding and decoding circuits of these codes employ multibit linear feedback shift register (LFSR) which, if used in a DRAM chip, will introduce very high access delay. By concatenating *finite projective geometry codes* (PGC) [23], a linear code can be constructed that can correct double-bit soft errors in a memory chip [25]. The main problem with this coding technique is that it divides an n-bit memory into $n^{1/3}$ subarrays, i.e., a 16-Mb DRAM will be organized into 256 subarrays (as discussed in Appendix A). This is an unrealistic scheme because it will introduce very high decoder routing complexity, and also it will increase the chip area considerably. The practical 16-Mb DRAM manufactured by NTT employs only 8 partitions. Another problem with this code is that it can correct only two errors in the entire memory, and cannot correct faults such as a bit line that is stuck-at or short/open because of its defective sense amplifier.

An efficient code is proposed in this section that can be easily implemented within the framework of high-density memories. One of the constraints on the code is that its encoding and decoding circuit should be compatible to the low intercellular pitch width. A rectangular product code, which can correct only a single-bit error, is known to satisfy these constraints, and many commercial chip manufacturers have successfully integrated the ECC based on product code in 4-Mb and 16-Mb DRAM chips [26]. In this section, a coding scheme, called here an *augmented product code,* is constructed by adding a set of diagonal parity bits to the conventional product code, which uses vertical and horizontal parity bits. Like the product code, the proposed code has simple encoding and decoding circuits that match the pitch width constraint of a DRAM chip, it automatically corrects the selected cell if it is faulty, and it generates error flags for diagnosing the second faulty bit, in case a double-bit soft error occurs. The rest of the section describes an effective way of organizing the product code in each word line within the DRAM chip and then how to construct an augmented product code from the rectangular organization of the product code.

[1] The time between the occurrence of a fault and its manifestation as a malfunction is defined as the fault latency.

A. The Conventional Rectangular Product Code

Let a DRAM chip with $n = (sm^2)$ information or data bits be organized into s subarrays each of size $m_1 \times m_2$ (i.e., each subarray has m_2 memory cells in each of its m_1 word lines). The DRAM is called nonredundant if $m_1 m_2 = m^2$. In a fault-tolerant DRAM chip usually $m_1 m_2 > m^2$. In order to construct a product code for each word line, m_2 cells are organized in the form of a logical rectangular array of size $(p+1) \times (q+1)$ where m information bits describe the inner array $p \times q$ such that $p \cdot q = m$, and the $(p+1)$th (bottom-most) row and the $(q+1)$th (rightmost) column consist of parity bits (as shown in Fig. 4). Let $C_{u,v}^k$ be the memory cell located at the crosspoint of the uth word-line and the vth bit line in the kth subarray, and $c_{u,v}^k$ be the content of the cell $C_{u,v}^k$.

Definition 1: For all $0 \leq u \leq m_1 - 1$ and $0 \leq k \leq s-1$, the string of binary bits $\{c_{u,0}^k, c_{u,1}^k, \cdots, c_{u,m_2-1}^k\}$ forms a product codeword if m_2 bits are organized into a $(p+1) \times (q+1)$ rectangular array such that if $b(i,j) = c_{u,iq+j}^k$ then for all $0 \leq i \leq p-1, b(i,q) = \oplus\Sigma_{s=0}^{q-1} b(i,s)$ and for all $0 \leq j \leq q-1, b(p,j) = \oplus\Sigma_{r=0}^{p-1} b(r,j)$. The Kronecker product[2] of $C_1 \otimes C_2$, where for all $i, C_1 = \{b(i,0), b(i,1), \cdots, b(i,q)\}$ and for all $j, C_2 = \{b(0,j), b(1,j), \cdots, b(p,j)\}$ defines a product code, $\mathrm{PC}(pq, p+q+1)$, where $pq = m$ and $m_2 - m = p+q+1$.

In the above definition, for all $0 \leq i \leq p-1$, the cell $b(i,q)$ is the ith horizontal parity bit in the rectangular array. Similarly, for all $0 \leq j \leq q-1$, the cell $b(p,j)$ is the jth vertical parity bit in the rectangular array. The ratio of the number of redundant bits to the total number of bits in the codeword is called the coding efficiency. If $p \approx q$, the coding efficiency for the product code will be maximized to $(2p/(p+1)^2) + 1/(p+1)^2 \approx 2/\sqrt{m}$. Since $m \approx \sqrt{n/s}$ and there are $\sqrt{n/s}$ product codes in each of s subarrays, in a DRAM chip with n information bits, altogether $2(sn^3)^{1/4}$ bits are required to correct as many as $sn^{1/2}$ single-bit/word-line errors. For example, in a 16-Mb DRAM organized into 16 subarrays, altogether $1M$ redundant cells (6.6%) are required to correct up to 16 384 single-bit/word-line soft errors.

In order to read a cell $C_{u,iq+j}^k$, the following two equalities are checked by parity generators: $\oplus\Sigma_{r=0}^{p} b(r,j) = 0$ and $\oplus\Sigma_{s=0}^{q} b(i,s) = 0$. If both the equalities are not true, then a single-bit error is corrected by complementing $c_{u,iq+j}^k$. If only the first equality is not true, then for $0 \leq i \leq p-1$ the second

[2] The Kronecker product, also called the direct or outer product, of two matrices $A = [a_{ij}]$ of size $m \times n$ and $B = [b_{kl}]$ of size $p \times q$ is defined as an $mp \times nq$ matrix $C = A \otimes B = [c_{rt}] = [a_{ij} b_{kl}]$.

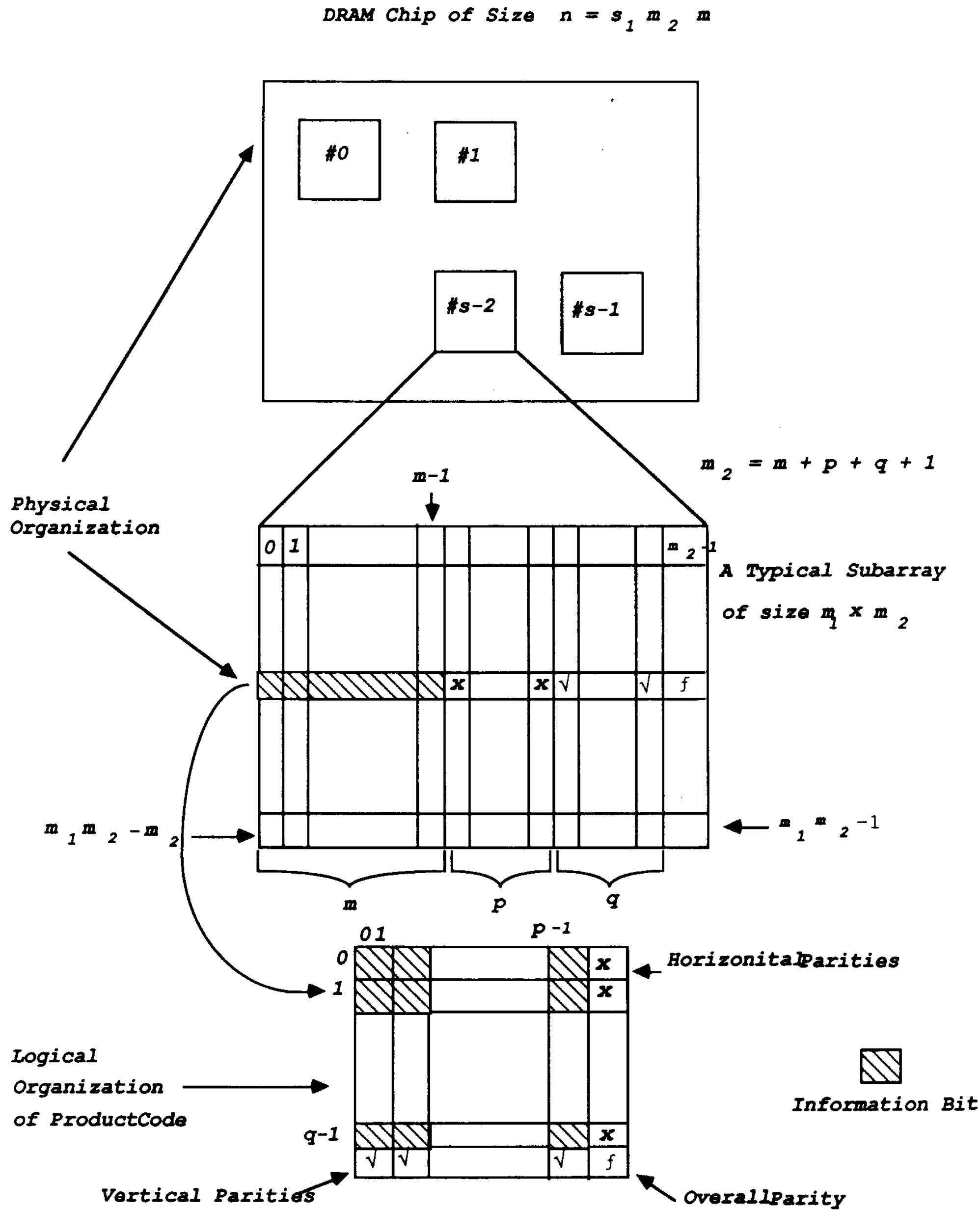


Fig. 4. Typical fault-tolerant DRAM chip using product code.

equality is tested; if the second equality is not true for $i = t$, then $c^k_{u,tq+j}$ is complemented to correct the single-bit error. Similarly, if only the second equality is not true, then the first equality is tested for $0 \le j \le q-1$; $c^k_{u,iq+t}$ is complemented if the first equality is not true for $j = t$. Thus all single-bit errors can be corrected. If there are two errors (say, cells $C^k_{u,lq+j}$ and $C^k_{u,iq+r}$) then the above scheme will incorrectly complement the cell $C^k_{u,iq+j}$. Hence, the product code fails to perform if a double-bit memory-cell upset occurs. In order to write a data on the cell $C^k_{u,iq+j}$, its content $c^k_{u,iq+j}$ is checked by a read operation, and it is compared with the input data. If the result of the comparison is equivalent, the content of the cell remains unchanged. If the result of the comparison is different, the contents of cells $C^k_{u,iq+j}$, $C^k_{u,pq+j}$, $C^k_{u,iq+q}$ and $C^k_{u,pq+p+q}$ are complemented.

A typical implementation of a product code, PC(9, 7) is shown in Fig. 5 where to each word line, having 9 information bits, 7 parity bits are added so that the overall word-line size is 16 bits. These 16 bits are organized into a logical 4×4 array. The parity bit of the ith row is stored in the cell denoted by H_i, and the parity bit of jth column is stored in the cell denoted by V_j. The overall parity of the word-line is stored

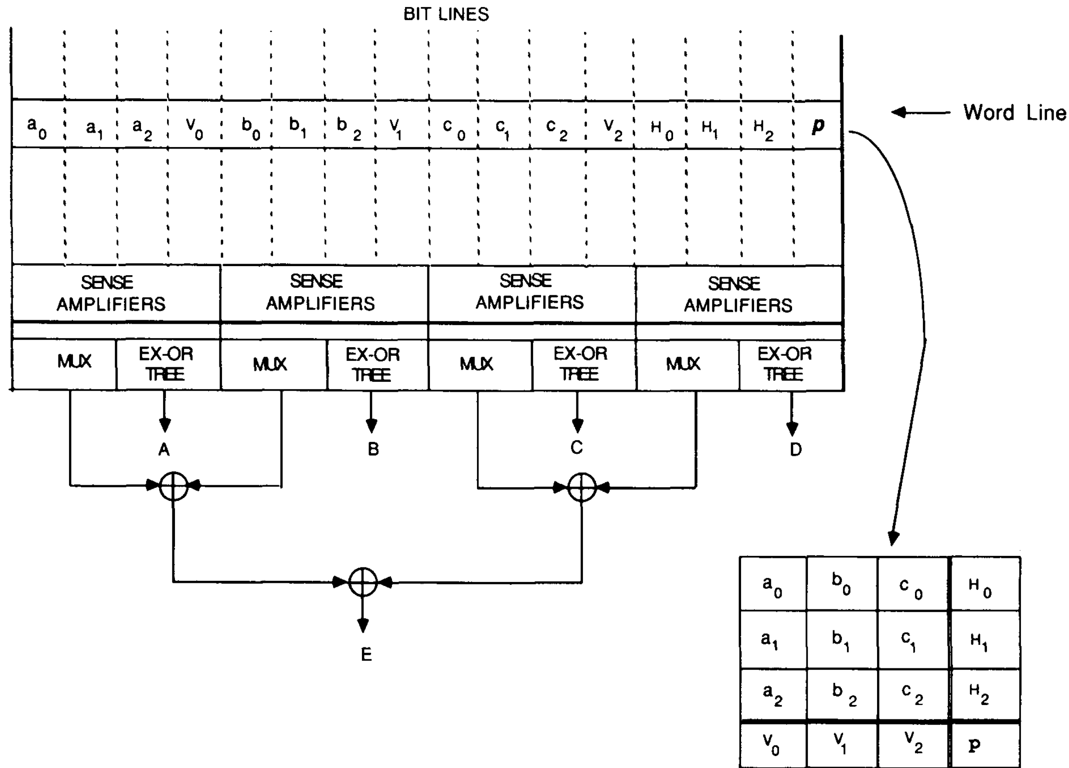


Fig. 5. Product code (9, 7) organization in a DRAM.

in the cell designated by π. Altogether, five parity trees and four multiplexers are used to construct the triplet from which the corrected bit can be read. In general, for a square array with a codeword of m bits, altogether $\sqrt{m}+2$ parity trees each of height $0.5\log m$, and $\sqrt{m}+1$ multiplexers of size $\sqrt{m}+1{:}1$ are needed.

B. The Proposed Code: An Augmented Product Code

In order to correct double errors in a codeword, an *augmented product code* (APC) is constructed by adding a set of p diagonal parity bits to the rectangular $p+q+1$ parity bits in the product code. It must be emphasized that the proposed code can detect and correct all single- and double-bit errors in the parity bits and information bits from the error syndrome represented as a quadruplet $\langle X(i), Y(j), D(t), \Pi\rangle$, where the symbols represent error flags for horizontal, vertical, diagonal, and the overall groups of parities and information bits, and will be explained later.

Definition 2: For all $0 \le u \le m_1 - 1$ and $0 \le k \le s-1$, the string of binary bits $\{c^k_{u,0}, c^k_{u,1}, \cdots, c^k_{u,m_2-1}\}$ forms an augmented product codeword if m_2 bits are organized into a $p \times q$ rectangular array of information bits and a linear array of $2p+q+1$ parity check bits such that if $b(i,j) = c^k_{u,iq+j}$ and $l(t) = c^k_{u,pq+t}$, then for all $0 \le i \le p-1, l(i) = \oplus\Sigma_{s=0}^{q-1} b(i,s)$; for all $0 \le j \le q-1, l(p+j) = \oplus\Sigma_{r=0}^{p-1} b(r,j)$; for all $0 \le i \le p-1, 0 \le j \le q-1, l(p+q+t) = \oplus\Sigma_{s=0}^{q-1} b((t+s) \bmod p, s)$, where $t = (p+i-j \bmod p) \bmod p$. The Kronecker product of $C_1 \otimes C_2$, where for all $i, C_1 = \{b(i,0), b(i,1), \cdots, b(i,q-1), l(i)\}$ and for all $j, C_2 = \{b(0,j), b(1,j), \cdots, b(p-1,j), l(p+j)\}$ along with $l(p+q+t)$ for all i and j, defines an augmented product code, $\mathrm{APC}(pq, 2p+q+1)$, where $pq = m$ and $m_2 - m = 2p+q+1$.

Clearly in the above definition, for all $0 \le i \le q-1$, the cell corresponding to $l(i)$ contains the ith horizontal parity bit; for all $0 \le j \le q-1$, the cell corresponding to $l(p+j)$ contains the jth vertical parity bit; for all $0 \le i \le p-1, 0 \le j \le q-1$, the cell corresponding to $l(p+q+t)$ contains the tth diagonal parity bit, where $t = (p+i-j \bmod p) \bmod p$. The entire m_2 bits in a DRAM word line are organized into a $p \times q$ array of information or data bits, p bits of horizontal parity, p bits of diagonal parity, q bits of vertical parity, and one bit of overall parity, such that $m_2 = pq + 2p + q + 1$. The overall parity bit π is computed over pq information bits as $\pi = \oplus\Sigma_0^{p-1}\Sigma_0^{q-1} b(i,j)$. The diagonal parity bit corresponding to the data bit $b(i,j)$ is computed by reading out the data bits in a cyclic path given by

$$
\begin{aligned}
b(i,j) &\to b(i+1, j+1), && \text{if } j < p-1, i < q-1,\\
b(i,p-1) &\to b(i,0), && \text{if } j = p-1,\\
b(q-1,j) &\to b(0,j), && \text{if } i = q-1.
\end{aligned}
$$

For an n-bit DRAM organized into s square subarrays, the pro-

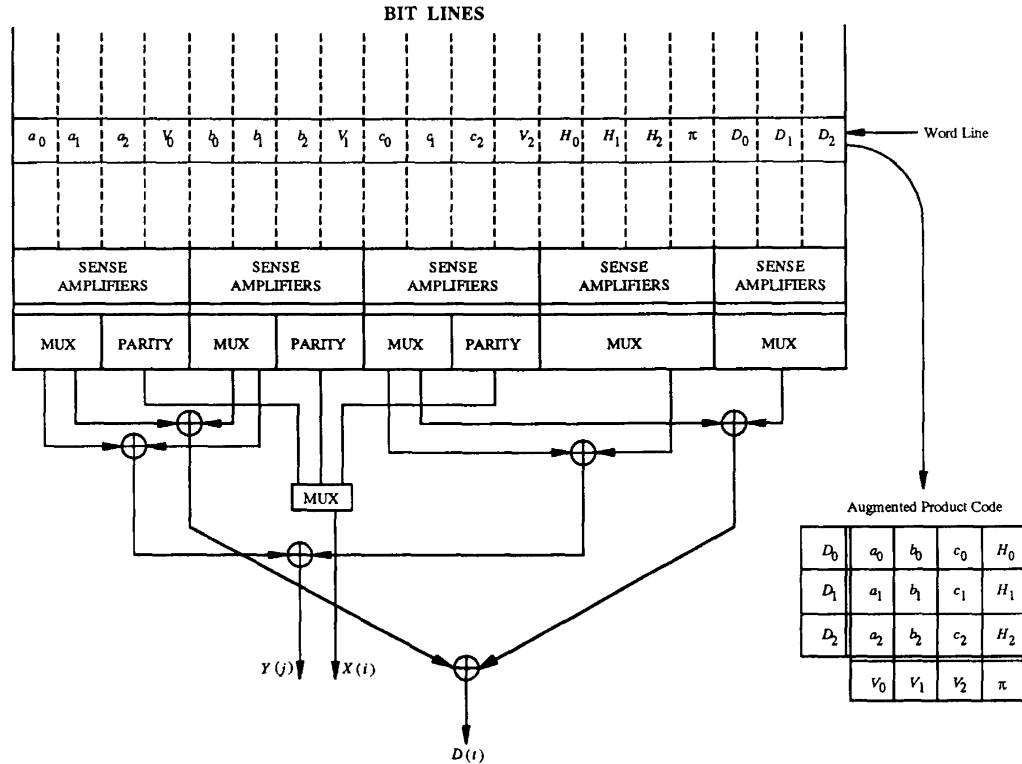


Fig. 6. Augmented product code APC (9, 10) organization in a DRAM.

posed coding will need about $3(sn^3)^{1/4} + (sn)^{1/2}$ redundant bits and will have approximately $3(s/n)^{1/4}$ coding efficiency. Thus a 16-Mb DRAM organized into 16 subarrays will need about 9% extra redundant bits.

In order to read the cell $C^k_{u.iq+j}$ (i.e., $b(i,j)$), the values $c^k_{u.pq+i}, c^k_{u.pq+p+j}, c^k_{u.pq+p+q+t}$ (where $t = (p + i - j \bmod p) \bmod p$) and $c^k_{u.pq+2p+q}$ are compared with $l(i), l(p+j), l(p+q+t)$ and π, respectively. Let $X(i), Y(j), D(t)$ and Π, respectively be the results of comparisons. In order to write a data on the cell $C^k_{u.iq+j}$ (i.e., $b(i,j)$), the value of $c^k_{u.iq+j}$ is compared with the data. If they are different, then the values $c^k_{u.pq+i}, c^k_{u.pq+p+j}, c^k_{u.pq+p+q+t}$ (where $t = (p+i-j \bmod p) \bmod p$) and $c^k_{u.pq+2p+q}$ corresponding to the parity bits $X(i), Y(j), D(t)$ and Π, respectively, are complemented. Since there are four parity check bits corresponding to each data bit, the $\mathrm{APC}(pq, 2p+q+1)$ is a distance five code, and can detect and/or correct all double-bit errors in the codeword, including the parity bits. The different error syndromes have been represented by Tabel I, and it can be seen that double errors can be corrected from the error patterns.

IV. An Implementation of the Augmented Product Code in a DRAM Chip

The APC can be easily implemented within a DRAM chip. A typical implementation of a DRAM chip with m-bit-wide bit-lines will contain $3\sqrt{m} + 1$ redundant bits per bit line. A DRAM where each word line has 9 data bits (organized into 3×3 matrix) and 10 parity bits is shown in Fig. 6. The distribution of horizontal, vertical, and diagonal parities on the word line is also shown. The vertical parities are easily computed by the parity trees (shown by PARITY in the diagram). The horizontal and diagonal parities are computed by the external EX-OR gates shown in Fig. 5. The suitable bit is selected by the MUXes. If there is no error in the selected bit-line, all the outputs $X(i), Y(j)$ and $D(t)$ are 0's. Clearly, from Table I it is evident that the different error lines can be expressed as Boolean functions of parity bits as shown below:

1) Single Error = $\Pi[X(i)Y(j)D(t) + \overline{Y}(j)\overline{D}(t) + \overline{X}(i)\overline{D}(t) + \overline{X}(i)\overline{Y}(j)]$
2) Double Error = $\overline{\Pi}[X(i) + Y(j) + D(t)]$
3) $b(i,j)$-Erroneous = $X(i)Y(j)D(t)$
4) Fatal Error = $\overline{\Pi}[X(i)Y(j)\overline{D}(t) + X(i)\overline{Y}(j)D(t) + \overline{X}(i)Y(j)D(t)]$
5) $b(\bar{i},j)$-Erroneous = $\Pi\overline{X}(i)Y(j)\overline{D}(t)$
6) $b(\bar{i},\bar{j})$-Erroneous = $\Pi\overline{X}(i)\overline{Y}(j)D(t)$
7) $b(i,\bar{j})$-Erroneous = $\Pi X(i)\overline{Y}(j)\overline{D}(t)$

$$t = (p + i - j \bmod p) \bmod p.$$

Fig. 7 shows a circuit that can automatically correct all single-bit errors. If a single-bit or double-bit error occurs, and $b(i,j)$-Erroneous signal is high, the bit $b(i,j)$ is automatically corrected and written back. The Fatal Error flag indicates

TABLE I
ALL POSSIBLE PATTERNS IN CODING

Π	$X(i)$	$Y(j)$	$D(t)$	Remarks
No Error	No Error	No Error	No Error	No Error $b(i,j)$ Error-Free
Error	Error	Error	Error	Single Error, $b(i,j)$ Erroneous
Error	Error	No Error	No Error	Single Error, $b(i,j)$ Error-free, $b(i,\bar{j})$ Erroneous
Error	No Error	Error	No Error	Single Error, $b(i,j)$ Error-free, $b(\bar{i},j)$ Erroneous
Error	No Error	No Error	Error	Single Error, $b(i,j)$ Error-free
Error	No Error	No Error	No Error	Single Error, $b(i,j)$ Error-free, $b(\bar{i},\bar{j})$ Erroneous
No Error	Error	No Error	Error	Double Error, $b(i,j)$ & $b(\bar{i},j)$ are Erroneous or, $b(i,\bar{j})$ & $b(\bar{i},\bar{j})$ Erroneous
No Error	No Error	Error	Error	Double Error, $b(i,j)$ & $b(i,\bar{j})$ Erroneous or, $b(\bar{i},j)$ & $b(i,\bar{j})$ Erroneous
No Error	Error	Error	No Error	Double Error, $b(i,j)$ & $b(\bar{i},\bar{j})$ Erroneous or, $b(\bar{i},j)$ & $b(i,\bar{j})$ Erroneous
No Error	Error	No Error	No Error	Double Error, $b(i,j)$ is not Erroneous
No Error	No Error	Error	No Error	Double Error, $b(i,j)$ is not Erroneous
No Error	No Error	No Error	Error	Double Error, $b(i,j)$ is not Erroneous
No Error	Error	Error	Error	Double Error, $b(i,j)$ is Erroneous

$t = (p+i-j \bmod p) \bmod p$

a double-bit error has occurred, and in the event of such a double error, all the bits in the vertical column of the augmented product code are read to determine whether the selected bit $b(i,j)$ is faulty. If the selected bit is faulty, it is complemented and rewritten by a separate hardware. In case a single-bit error occurs where the selected bit is not faulty, it can be easily corrected by identifying the signals for $b(i,\bar{j})$-Erroneous, $b(\bar{i},\bar{j})$-Erroneous and $b(\bar{i},j)$-Erroneous. It may be noted that the symbol $b(i,\bar{j})$-Erroneous denotes that a cell on the ith row of the product code and on the jth column is faulty where $0 \le \bar{j} \ne j \le q-1$. Similarly, the symbol $b(\bar{i},j)$-Erroneous denotes that a cell on the jth column of the product code and on the $\bar{i}$th row is faulty where $0 \le \bar{i} \ne i \le p-1$. The symbol $b(\bar{i},\bar{j})$-Erroneous dentoes that a cell on the diagonal line of the cell whose location is on the ith row and jth column of the product code is faulty; the faulty cell is located on the crosspoint of the $\bar{i}$th row and $\bar{j}$th column of the product cod where $0 \le \bar{i} \ne i \le p-1$ and $0 \le \bar{j} \ne j \le q-1$. These error flags reduce the search time to locate the faulty bits. In a 64-Mb DRAM with $m_2 = 4084$, it requires at most 16 read operations to locate the faulty cell by monitoring these flag bits. This improves the availability of the RAM chip. The following are the algorithms for read and write operations:

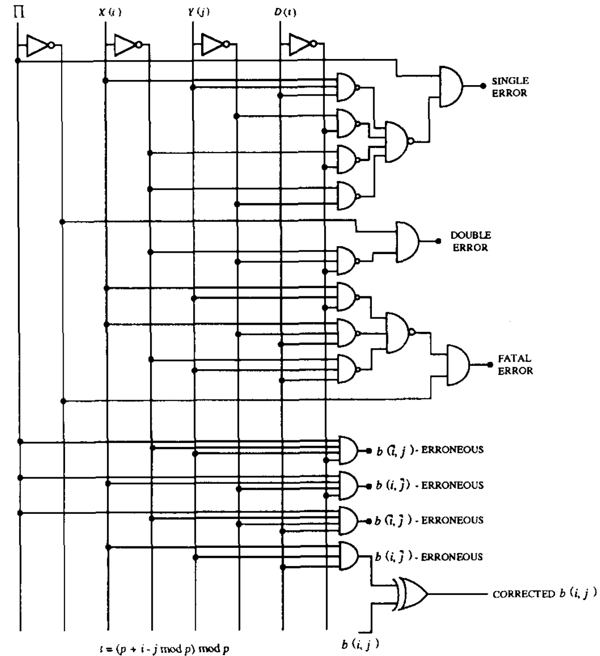


Fig. 7. Error-correcting circuit using augmented product code.

P1: Procedure for Read Operation

1. Read Cell $b(i,j)$;
2. Check the Error Lines;
3. If Single Error occurs
4.   If $b(i,j)$-Erroneous is high
5.   Correct the cell;
6.   If $b(\bar{i},j)$ -Erroneous is high
7.   Defective cell on the same row;
8.   Locate and correct it;
9.   If $b(i,\bar{j})$-Erroneous is high
10.   Defective cell on the same column;
11.   Locate and correct it;
12.   If $b(\bar{i},\bar{j})$-Erroneous is high
13.   Defective cell on the same diagonal line;
14.   Locate and correct it.
15. If Double Error occurs
16.   If Fatal Error = 0,
17.   Correct the cell;
18.   If Fatal Error = 1,
19.   Diagnose whether $b(i,j)$ is Erroneous;
      Correct the cell if it is erroneous.

P2: Procedure for Write Operation

1. Use P1 (above Procedure for Read Operation) to read the cell;
2. If the data to be written is same as $b(i,j)$
3.   Write back $b(i,j)$;
4. If the data to be written is the complement of $b(i,j)$
5.   Complement $b(i,j)$, $X(i)$, $Y(j)$, $D(t)$, and Π.

V. Area and Timing Overhead of the APC

One of the primary concerns the memory manufacturers have is that on-chip ECC tends to reduce the memory cycle time and requires extra silicon area, in addition to the fact that the ECC circuit introduces severe layout problem since it is less regular than the memory plane. In order to justify the use of on-chip ECC circuit, it is necessary to examine the overhead in silicon area and timing, as opposed to the payoffs in increased reliabilty. In this section a detailed analysis is done to estimate the area and timing overhead, and in the next section the improvements in soft-error rate (SER) and mean-time to failure (MTTF) are calculated to show the effectiveness of the proposed ECC.

A. Area Overhead

The proposed ECC circuit consists of multiplexers, parity trees, additional sense amplifiers and encoders. Assume that the k information bits per word line in a DRAM chip are organized into a $\sqrt{k} \times \sqrt{k}$ array with additional $3\sqrt{k}+1$ bits corresponding to the horizontal, vertical, and diagonal parity bits, and an overall parity bit. Each word line, therefore, consists of $(k + 3\sqrt{k} + 1)$ bits, which can be grouped into three categories as follows.

1) *Groups of category G1:* There are $\sqrt{k}$ members in this group, where each member has $(\sqrt{k}+1)$ bits corresponding to each ith column of the logical array in APC with $\sqrt{k}$ information bits and one bit for storing V_i.
2) *Groups of category G2:* There is one group of $(\sqrt{k}+1)$ bits which consists of $\sqrt{k}$ bits corresponding to the horizontal parities (H_i's) and one bit for storing Π.
3) *Groups of category G3:* There is one group of $\sqrt{k}$ bits for storing the different D_i bits.

It may be noted that $2\sqrt{k}$ multiplexers are required for category G1, and one each for categories G2 and G3, and also for calculating X as shown in Fig. 8. Similarly, altogether $\sqrt{k}+2$ parity trees are required to calculate the parity bits—$\sqrt{k}$ for category G1, and one each for calculating Y and D (none for G2 and G3).

It may be noted that each multiplexer is a $(\sqrt{k}+1)$-to-1 type which can be constructed hierarchically by using only 2-to-1 muxes, each occupying an area of a_m unit. If a multiplexer tree is built using 2-to-1 mutliplexers, then the area A_m of each $(\sqrt{k}+1)$-to-1 mux is given by

$$A_m = ((\sqrt{k}+1)/2) \cdot [\log(\sqrt{k}+1)/2] \cdot a_m.$$

Similarly, if the area of one EX-OR gate is a_x, then the area, A_x, required by one EX-OR parity tree with $(\sqrt{k}+1)/2$ input bits, is given by

$$A_x = ((\sqrt{k}+1)/2) \cdot [\log(\sqrt{k}+1)/2] \cdot a_x.$$

The total area needed by multiplexer trees and EX-OR parity trees is

$$[(2\sqrt{k}+3)a_m + (\sqrt{k}+2)a_k]((\sqrt{k}+1)/2)[\log(\sqrt{k}+1)/2].$$

In addition to muxes and parity trees, additional memory cells for parity bits and the associated sense amplifiers will be used in the ECC circuits. Since there are k bits of information per word, the number of extra memory cells required is $k_w(3\sqrt{k}+1)$, where k_w is the number of word lines in the memory array. If the area required by one memory cell is a_{mem}, the total area required by the extra memory cells used in $k_w \cdot (3\sqrt{k}+1)a_{\text{mem}}$. Also, the number of extra sense amplifiers required is $(3\sqrt{k}+1)$. If the area required by one sense amplifier is a_s, the extra area required by the sense amplifiers is $(3\sqrt{k}+1)a_s$. In addition to the circuitry discussed above, two encoders are required for encoding $(\sqrt{k}+1)$ bits into $\log(\sqrt{k}+1)$ bits, which locates the bit line containing an erroneous bit, if a double bit-error occurs. Each of these encoders can be assumed to have an area of $(\sqrt{k}+1)[\log(\sqrt{k}+1)]a_{\text{enc}}$.

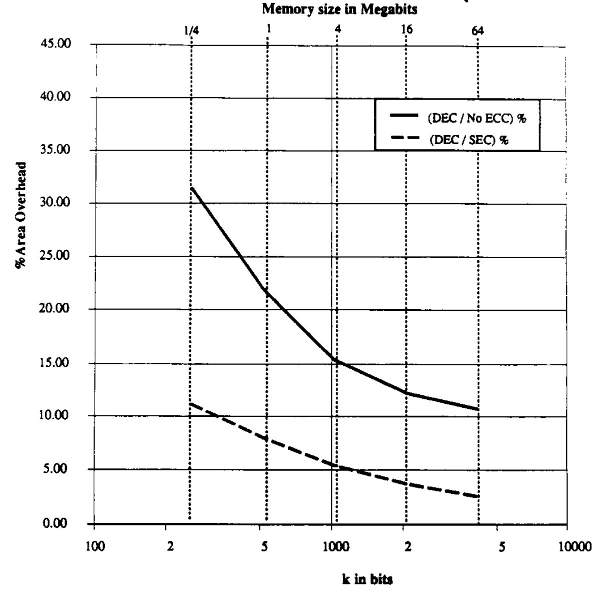


Fig. 8. Area overhead for the APC.

Some of the factors that have not been considered in the above discussion are the increase in the area of the bit-line decoder. Since the number of bits per word has increased, the area of the bit-line decoder is also greater. But this increase in area is not very significant. Thus, the total increase in the area denoted by $A_{\text{over-head}}$ is given by the sum total of the increase in area due to the various factors mentioned above:

$$\begin{aligned} A_{\text{over-head}} = {} & [(2\sqrt{k}+3)a_m + (\sqrt{k}+2)a_x)] \\ & \cdot ((\sqrt{k}+1)/2)[\log(\sqrt{k}+1)/2] \\ & + k_w \cdot (3\sqrt{k}+1)a_{\text{mem}} + (3\sqrt{k}+1)a_s \\ & + (\sqrt{k}+1)[\log(\sqrt{k}+1)] \cdot a_{\text{enc}}. \end{aligned}$$

The area required by a DRAM of k_w words of k bits of information each without any error correcting circuitry is given by

$$A_0 = k_w \cdot k \cdot a_m + k \cdot a_s.$$

In both expressions given above the area required by the row and column decoding circuitry should be added.

Using the above equations, the area overhead for DRAM chips with ECC that can correct double errors have been calculated for different chip sizes. In Fig. 8, the overhead of a DRAM with the proposed ECC are compared with chips with no ECC and with SEC-type ECC. It can be seen that the proposed DEC-type code requires about twice the area than the SEC-type product code when the chip size is $4M$ or more, but this can be justified by the improvement (over SEC) in storage reliability by an order of magnitude, as discussed in Section VI-A. For a 16-Mb DRAM chip with 16 partitions, the chip area overhead due to the proposed augmented code is about 8% of the overall chip area. This is somewhat close to the values obtained by Yamada [3], who designed 16-Mb DRAM's with on-chip single-error correcting circuit. The ECC layout should be carefully done to optimize the DRAM access delay and ECC area. Acces delay can be reduced by designing a selector-merged ECC circuit where transmission parity checkers and selectors, which select data from cells belonging to the appropriate parity groups, are arranged in column circuits without long bus lines.

B. Timing Overhead

The presence of ECC and the procedures P1 and P2 increase the length of average memory cycle. The various steps in reading a cell and the corresponding time delays involved are given below.

Procedure Read:

1) Decode the bit line—$t_{\text{decode-bit}}$.
2) Precharge bit line—$t_{\text{precharge}}$.
3) Decode the word line—$t_{\text{decode-word}}$.
4) Enable the word line—$t_{\text{word-enable}}$.
5) Read the selected cell—$t_{\text{read-cell}}$.
6) Check the error signals to see if a single or a double error has occurred. The computation of error values requires a delay $t_{\text{mux}} + t_{\text{xor-tree}} + 4t_{\text{gate}}$ where t_{mux} is the delay involved in passing the signal through a multiplexer and $t_{\text{xor-tree}}$ is the delay involved in passing the signal through the EXOR trees, and $4t_{\text{gate}}$ is the delay corresponding to the circuit in Fig. XX.

Assume that τ_r is the time required for reading a cell using the procedure READ. If w is the probability of a write operation, then the average memory cycle time will be given by $\bar{t}_{\text{ECC}} = \tau_r + wt_{\text{write-cell}}$, where

$$\tau_r = (t_{\text{decode-bit}} + t_{\text{precharge}} + t_{\text{decode-word}} + t_{\text{word-enable}}) + t_{\text{read-cell}} + t_{\text{mux}} + t_{\text{xor-tree}} + r_{\text{gate}}.$$

The average memory cycle time required for a memory without any error correction circuitry is

$$\bar{t}_0 = (t_{\text{decode-bit}} + t_{\text{decode-word}} + t_{\text{word-enable}}) + (1-w)(t_{\text{precharge}} + t_{\text{read-cell}}) + w \cdot t_{\text{write-cell}}.$$

The detailed computations of the time required for various steps in reading a cell are shown below.

1) Delay in decoding word-line address $t_{\text{decode-word}}$:

$$t_{\text{decode-word}} = k_s \tau f \log_f(k_w) + 1.02 R_a C_a$$

where k_s is the structure constant with value 2.5 for inverter ratio 4, k_w is the number of words, τ is a gate delay, f is the stage factor for the driving inverters, R_a and C_a represent the resistance and capacitance across the address decoder for word line decoding, respectively.

2) Delay in decoding a word line $t_{\text{decode-bit}}$:

$$t_{\text{decode-bit}} = k_s \tau f \log_f(k_b) + 1.02 R_c C_c$$

where k_b is the number of bit lines, R_c and C_c represent the resistance and capacitance, respectively, across the address decoder for bit-line decoding.

3) Delay in precharging a bit line $t_{\text{precharge}}$:

$$t_{\text{precharge}} = R_o C_{\text{bit-line}}$$

where R_o is the output resistance of the driver for bit line and $C_{\text{bit-line}}$ is the capacitance of the bit line. This delay is very small.

4) Word-line enable delay $t_{\text{word-enable}}$:

$$t_{\text{word-enable}} = 1.02 R_w C_w + 2.2 r_{tw} C_w$$

where R_w and C_w represent the resistance and the capacitance of the word line and r_w is the resistance of the transistor driving the word line.

5) Delay in reading a selected cell $t_{\text{read-cell}}$:

$$t_{\text{read-cell}} = R_t(C_{\text{cell}} + C_{\text{bit-line}})$$

where R_t is the resistance of the transistor in a memory cell through which the C_{cell} is charged. Other terms are as explained above.

6) Delay in writing into a cell $t_{\text{write-cell}}$:

$$t_{\text{write-cell}} = (R_t + R_o)(C_{\text{cell}} + C_{\text{bit-line}}).$$

The above expression is the same as that for $t_{\text{read-cell}}$ except for the output resistance of the driver for a bit line R_o being added to the resistance.

7) The delay in passing through a multiplexer t_{mux}

$$t_{\text{mux}} = t_m \log_2 [(\sqrt{k} + 1)/2]$$

where t_m is the delay through a 2-to-1 multiplexer, which is equivalent to the delay through two NAND gates. k is the number of information bits per word and k_b, and the number of bit lines is given by $k_b = k + 3 \cdot \sqrt{k} + 1$.

8) The delay in passing through the EX-OR parity tree $t_{\text{xor-tree}}$

$$t_{\text{xor-tree}} = t_x \log_2 [(\sqrt{k} + 1)/2]$$

where t_x is the delay in passing through an EX-OR gate.

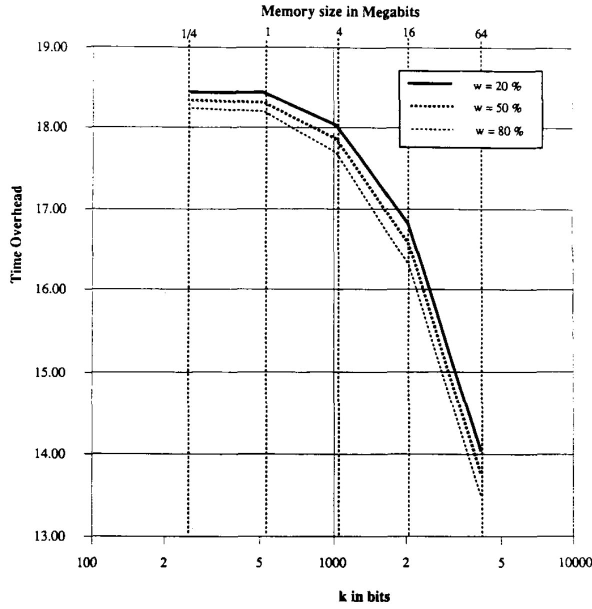


Fig. 9. Timing overhead for the APC compared with SEC.

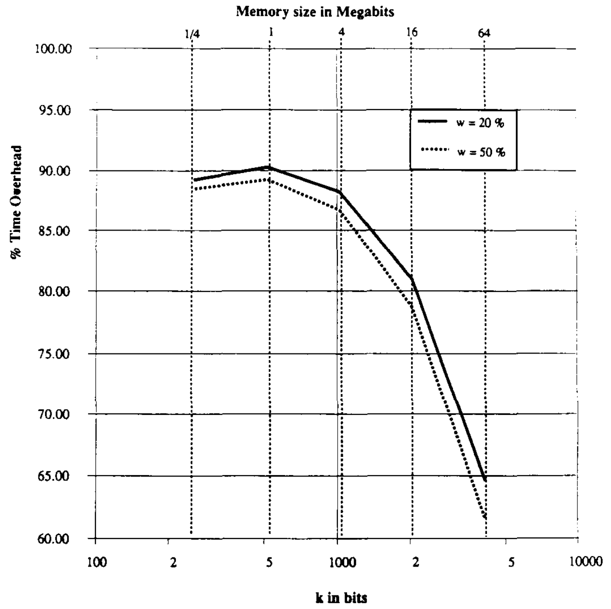


Fig. 10. Timing overhead for the APC compared with no ECC.

For different memory sizes, the timing overhead of the proposed ECC are compared in Fig. 9 with ECC that can correct a single error (SEC). For multimegabit DRAM's, this overhead is between 14% and 18% of the access time required by DRAM chips with SEC-type ECC, and it is virtually invariant of write probability, if $0.2 \leq w \leq 0.8$. The time overhead of the proposed ECC is compared with nonredundant DRAM's of various sizes. It may be noted that in the proposed architecture, the read-before-write operation and the additional ECC circuit increases the memory cycle time, but it is within 80% for a 16-Mb DRAM, as shown in Fig. 10. A typical delay of a 16-Mb DRAM with the proposed ECC was observed to be 63 ns as opposed to about 35 ns without any ECC.

VI. Reliability and Soft-Error Pattern Analysis

A. Reliability Modeling of the APC

In this section, the alpha-particle-induced soft-error rate (SER) in a DRAM chip with the proposed error-correcting mechanism has been analyzed and compared with that of a nonredundant DRAM without any error correction mechanism. In order to compute SER, the following assumptions are made.

Assumption 1: In practice, the intermittent faults in a DRAM result from alpha particles and sporadic process-related leakage currents that vary with temperature, static noise, noise pulses on power supply, data pattern in the memory, and so on. These faults manifest themselves randomly and can be represented by the Poissonian statistics with a mean rate of $\lambda = \lambda_\alpha + \lambda'$, where λ_α is caused by alpha particles, and λ' results from other intermittent faults. In a well-designed memory usually the soft errors are predominantely caused by alpha particles and, therefore, $\lambda_\alpha \gg \lambda'$.

Assumption 2: The alpha-particle-induced soft errors occur independently at the different word lines and at different subarrays within the chip.

Assumption 3: The alpha-particles flux density ϕ is uniform over the entire memory plane. Hence, $\lambda_\alpha = \phi(\sigma + \delta/2)^2$.

Assumption 4: The kinetic energy of the alpha particle is always sufficient to generate a single-bit error or double-bit soft errors. If the track of an alpha particle is limited within a single cell, a single-cell upset always occurs. If the track of an alpha particle permeates over two cells, or it hits between two cells, a double-cell upset occurs.

Let $p(t)$ be the probability that no soft error occurs in a memory cell within a time interval of $[0, t]$. The reliability of an $n(=sm^2)$-bit DRAM chip that can tolerate at most a double-bit failure in a word line, can be given by $R(t) = [p^m(t) + m(1-p(t))p^{m-1}(t) + 0.5m(m-1)p(t)^{m-2}(1-p(t))^2]^{sm}$, assuming that the DRAM chip is organized into s subarrays, each of size $m \times m$. From the above assumptions, the reliability function can be written as $R(\tau) \approx e^{-n\lambda\tau}[1 + m\lambda\tau + 0.5(m\lambda\tau)^2]^{sm}$. By using the notion of the hazard function $h(\tau) = \partial R(\tau)/R(\tau)$, the SER of the proposed error-correcting scheme can be shown to be $\mathrm{SER_{DEC}} = n\phi\psi - (\sqrt{ns}/\tau)(\log_e(1 + \sqrt{n/s}\phi\psi\tau + 0.5(n/s)(\phi\psi\tau)^2)$ since $\lambda \approx \phi(\sigma+\delta/2)^2 = \phi\psi$. For an n -bit DRAM without any error correction mechanism, the SER is given by $\mathrm{SER_0} = n\phi\psi$. Thus the error-rate improvement factor $\rho_{\mathrm{DEC}} = \mathrm{SER_0}/\mathrm{SER_{DEC}}$ is $1/(1 - (1/x)\log_e(1 + x + x^2))$, where $x = \sqrt{(n/s)}\phi\psi\tau$ and gives the average rate of failure of a word line in the DRAM. The MTTF of the memory chip with the proposed

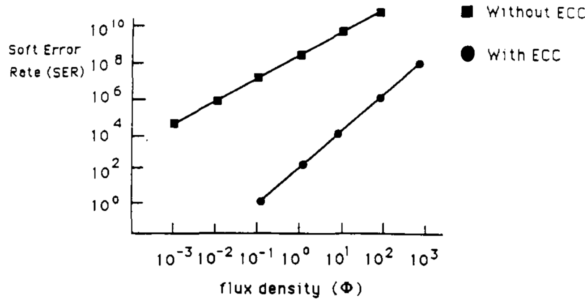


Fig. 11. Improvement in soft-error rate.

error-correcting scheme can be given by

$$\mathrm{MTTF}_{\mathrm{DEC}} = \int_0^{\infty} R(t)\,dt = \int_0^{\infty} e^{-n\lambda t}[1 + m\lambda t + 0.5(m\lambda t)^2]^m\,dt$$

where $\lambda = \psi\phi$ and $s = 1$. The integral is solved in Appendix B and from the result of integration, it can be found that $\mathrm{MTTF}_{\mathrm{DEC}} = (1/\psi\phi n)[6^{1/3}\Gamma(4/3)n^{1/3}] = 1.62/\psi\phi n^{2/3}$. The MTTF of the memory with the conventional SEC codes can be shown to be $\mathrm{MTTF}_{\mathrm{SEC}} = (1/\psi\phi n)[\sqrt{2}\Gamma(3/2)n^{1/4}] = 1.25/\psi\phi n^{3/4}$. Thus the proposed scheme improves the MTTF by a factor of $1.3n^{1/12}$, which for a 16 Mb memory will approximately result in 520% improvement in reliability. The proposed scheme is plotted for reliability (Fig. 11) and SER, and they are compared with a simplex system without any error-correcting mechanism. From the graph of SER vs. alpha-particle flux density shown in Fig. 11, it has been seen that the SER improvement factor is more than 10^6 for a square memory array of size 4 Mb when the alpha-particle flux density is $1/\mathrm{cm}^2/\mathrm{h}$.

B. Error-Pattern Analysis

In addition to correcting soft errors, the built-in ECC is capable of masking the fabrication-related hard faults and many reserchers [3] analyzed the manufacturing yield improvement by using ECC circuits. The resulting chip becomes partially nonredundant or fully nonredundant depending on how many hard faults are reconfigured during fabrication time. The potential problem of using such double-bit error-correcting circuits for improving the yield is that they cannot tolerate more than two hard errors per memory word line. Thus these error-correcting techniques are not adequate for common fabricational faults such as the word-line driver being faulty where the entire word line may be defective. The conventional row and column redundancy can be very effectively used to improve the yield for the fault-tolerant memory. The overhead associated with the error-correcting circuit is so high that it will be grossly underutilized if they are exclusively used to improve the yield. The resulting nonredundant DRAM will be intolerant of soft errors and their access time will be larger than the normal redundant DRAM.

It may be noted that the proposed error-correcting code can tolerate as many as two errors per word line. Thus in an n-bit DRAM organized into s square subarrays, the code can detect as many as $2\sqrt{sn}$ errors if no more than two errors occur per word line. Although in this paper we have considered up to two errors per word line, in high-density DRAM multiple-bit errors may occur with very low probability. By using Monte Carlo simulation Sai-Halasz *et al.* [15] showed that a large number of soft errors in the memory are one or two bits, and in a few instances a number of bits may be erroneous. It may be pointed out that the multiple errors occur randomly in the event of a cosmic shower, and the proposed error-correcting circuit will be able to correct the multiple soft errors as long as only two bits are erroneous in a word line. The probability of correcting the different multiple-bit soft errors can be estimated by combinatorial analysis as shown below.

Let the vector $\mathbf{P_m} = \langle p_0, p_1, \cdots, p_{m-1}\rangle$, where $m = \sqrt{n/s}$, denote the possible soft-error pattern in the DRAM. Each p_i in the pattern denotes the number of soft errors in a distinct row, such that if altogether k errors occur, then $k = \Sigma_{i=0}^{m-1} p_i$. Without any ambiguity it may be assumed that for all $0 \leq i \leq m-2, p_i \geq p_{i+1}$. Thus, if three-bit soft error occurs, then it is correctable by the proposed coding if the error patterns are $\langle 2, 1, 0, \cdots, 0\rangle$ or $\langle 1, 1, 1, \cdots, 0\rangle$. Hence, the number of ways that the first error pattern can occur is given by $\binom{m}{1}\binom{m-1}{1}\binom{m}{2}\binom{m}{1}$. The number of ways that the second error pattern can occur is given by $\binom{m}{1}\binom{m-1}{1}\binom{m-2}{1}\binom{m}{1}^3$. The total number of ways three soft errors may occur in

$$P_{\mathrm{DEC}}(3) = \frac{\binom{m}{1}\binom{m-1}{1}\binom{m}{2}\binom{m}{1} + \binom{m}{1}\binom{m-1}{1}\binom{m-2}{1}\binom{m}{1}^3}{\binom{m^2}{3}}$$

$$P_{\mathrm{DEC}}(4) = \frac{\binom{m}{1}\binom{m-1}{1}\binom{m}{2}^2 + \binom{m}{1}\binom{m-1}{1}\binom{m-2}{1}\binom{m}{2}\binom{m}{1}^2 + \binom{m}{1}\binom{m-1}{1}\binom{m-2}{1}\binom{m-3}{1}\binom{m}{1}^4}{\binom{m^2}{4}}$$

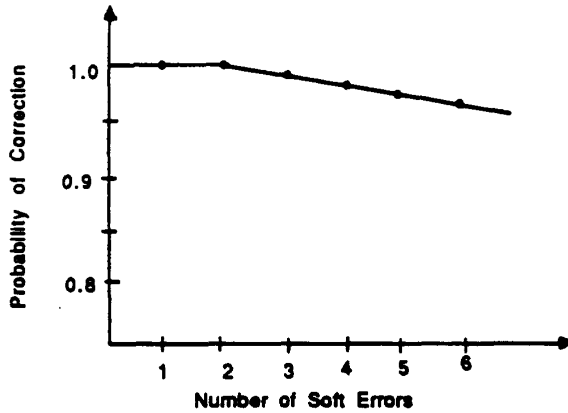


Fig. 12. Error correction probability in presence of multiple errors.

an m^2-bit DRAM is $\binom{m^2}{3}$. Hence the probability of correctable three-bit soft error is given in the first equation at the bottom of the page. Similarly, if four soft errors occur at random, they can be corrected if the error-bit patterns are one of these three: $\langle 2,2,0,0,\cdots,0\rangle, \langle 2,1,1,0,\cdots,0\rangle$ or $\langle 1,1,1,1,0,\cdots,0\rangle$. The probability of correctable four-bit error is also given in the second equation at the bottom of the page. Finally, in order to find the utility of the code, it is necessary to determine the probability that the code can correct all the errors knowing that at most k errors have occurred. For $k = 2\sqrt{n/s}(=2m,$ say), this probability can be shown to satisfy the following inequality:

$$P_{\mathrm{DEC}}(k = 2m) > \sum_{j=0}^{m-1} \frac{m^{m-j}\binom{m}{j}}{\binom{m^2}{m-j}} \times \sum_{j=0}^{m-1} \frac{(m-1)^{m-j}\binom{m-1}{j}}{\binom{m^2-m}{m-j}}$$

In Fig. 12 the probability that the proposed code will be able to detect multiple-bit soft errors has been plotted for different values of multiple-bit (k) errors.

VII. Conclusions

This paper discussed the problems of error correction in multimega bit dynamic random-access memory (DRAM) chips. A large number of alpha-particle-induced soft errors manifest themselves as double-bit errors when alpha particles hit the space between two vertically mounted trench capacitors. The conventional SEC/DED code, such as product and cubic codes, cannot correct these double-bit/word-line faults. A new augmented product code that employs diagonal parities in addition to horizontal and vertical parities is proposed in this paper to correct the double-bit errors. The proposed code has been compared with the projective geometry code (PGC), which can also correct two-bit errors (as discussed in Appendix A). But unlike the proposed code, the PGC cannot be easily implemented in a multimegabit memory. The PGC requires special decoders that compute over Galois Fields, and also is double-error correcting logic is very complex. The proposed code can tolerate up to $2\sqrt{sn}$ soft errors, and thereby it can correct the sense amplifier faults. In Table II, the proposed coding scheme is compared with the product code and the projective geometry code. An error-pattern analysis has been done to find out the probability of correcting multiple errors that may occur in a DRAM chip. In addition to detecting the on-line error, the error-coding circuit can be utilized to improve the fabrication yield. In a defective memory chip, the faulty cells can be automatically bypassed by the error-correction circuit and the resulting memory can be used as a nonredundant and nonfault-tolerant memory. But, in practice, it will be a better idea to employ a few extra rows and columns to bypass the fabrication defects, and to utilize the error-correcting circuit exclusively for correcting the field failures and soft errors.

TABLE II
Comparison of Three Coding Techniques

Criterion	Product Code	Projective Geometry	Proposed Code
Cell Redundancy	$2(s/n)^{1/4}$	$2/n^{1/3}$	$3(s/n)^{1/4}$
Code Distance	4	6	5
Error Correction	SEC	DEC/TED	DEC
Number of Faults	$\sqrt{sn}$ /chip	2/chip	$2\sqrt{sn}$ /chip
Sense Amp. Failure	Yes	No	Yes
Fault detection	Simple	Complex	Simple
Address Computation	Simple	Galois Field	Simple
Layout Complexity	Low	High	Medium
Decoding Time	Small	Large	Medium

Appendix A
Finite Projective Geometry Code

The finite projective geometry code is derived from the concept of projective geometry, which is obtained by adding a hyperplane at inifnity to Euclidean geometry [27].

Definition 4: An m-dimensional finite projective geometry $\mathrm{PG}(m,q)$ described over the elements of Galois Field $\mathrm{GF}(q)$[3] consists of a set of $(q^{m+1}-1)/(q-1)$ *points* together with a set of equal number of *lines* such that each line passes over $q+1$ points. Each point is denoted by nonzero $(m+1)$-tuples $(a_0, a_1, \cdots, a_m)$, where $a_i \in \mathrm{GF}(q)$ with the rule that $(a_0, \cdots, a_m)$ and $(\lambda a_0, \cdots, \lambda a_m)$ are the same point, where λ is a nonzero element of $\mathrm{GF}(q)$. A projective geometry code $\mathrm{PG}(m,q)$ is of length $q^m + q^{m-1} + \cdots + 1$ with symbols from $\mathrm{GF}(q)$.

Definition 5: An m-dimensional affine or Euclidean geometry, $\mathrm{EG}(m,q)$ is obtained by deleting the points of a fixed hyperplane (called the hyperplane at infinity) from the subspaces of a projective geometry. If the hyperplane is the line $[1 \quad 0 \quad \cdots \quad 0]$ consisting of all points with $a_0 = 0$, then q^m points of $\mathrm{EG}(m,q)$ can be labeled by the m-tuples $(a_1, \cdots, a_m)$, where $a_i \in \mathrm{GF}(q)$.

It may be noted that in block design, a projective plane is equivalent to a Steiner system $S(2, n+1, n^2+n+1)$, and

[3] A finite field with modulo q addition and multiplication is called a Galois Field (after the famous classical field theorist Evarist Galois) of q isomorphic elements if q is a power of a prime. For example, in GF(2), the operations correspond to binary EXCLUSIVE OR and AND.

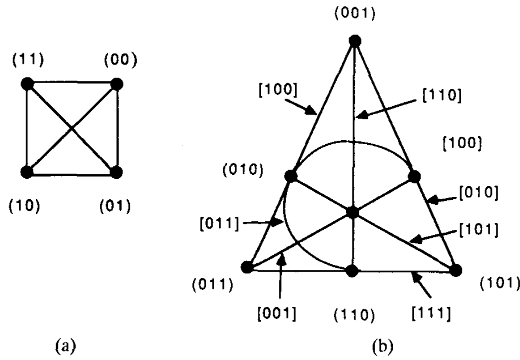


Fig. 13. Euclidean and projective geometries.

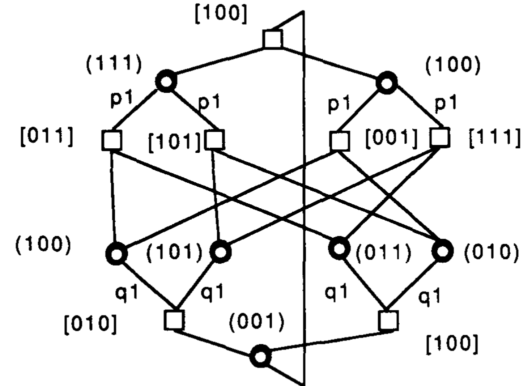


Fig. 14. Field-plane hexagon obtained from PG(2, 2).

an affine plane to an $S(2, n, n^2)$, where $n \geq 2$. Before it is explained how these geometries can be utilized to construct double-bit error-correcting codes, definitions 4 and 5 are illustrated below with an example. A two-dimensional Euclidean geometry EG (2, 2) consists of four points and six lines as shown in Fig. 13(a) and a projective geometry PG(2, 2) can be obtained by adding a hyperplane, the line [100] over the points (001), (011) and (010). The resulting projective geometry has seven points and seven lines, each line passing through three points as shown in Fig. 13(b). The point (001) intersects at infinity the parallel lines in the Euclidean geometry connecting the points (00) with (01) and (10) with (11). Similarly, the point (011) intersects at infinity the parallel lines connecting the points (10) with (01) and (00) with (11).

Definition 6: A linear concatenation of $q + 1$ projective geometry codes $\mathrm{PGC}(m, q)$ is defined as a set of $(q+1)(q^m + \cdots + 1) = q^{m+1} + 2q^m + \cdots + 1$ symbols from $\mathrm{GF}(q)$. The resulting code, denoted by FPH $(q^{m+1}, 2q^m + 2q^{m-1} + \cdots + 1)$, is constructed from the connectivity of $\mathrm{PG}(m, q)$. The projective geometry $\mathrm{PG}(m, q)$ is represented as a bipartite graph $\mathrm{G}(P, L, E)$, where P is the set of points in $\mathrm{PG}(m, q)$ and L is the set of lines in $\mathrm{PG}(m, q)$, and $E = \{e \in \langle p, l \rangle | p \in P, l \in L$ such that p lies on $l\}$. Each symbol in the codeword is a distinct edge in E of the graph G that is commonly known as *field-plane hexagon* [28].

For example, PG(2, 2) in Fig. 13(b) can be represented as a field-plane hexagon of diameter (for the maximum cycle) six as shown in Fig. 14. The set of lines and points in Fig. 13(b) have been represented by the square and circular nodes, respectively. All those points that belong to a line have been represented by the edges, and each node has degree three since in Fig. 13(b) each line passes through three points and also three lines meet at each point. In a two-dimensional projective geometry with q symbols $\mathrm{PG}(2, q)$ each node will have a degree of $q+1$ corresponding to $q+1$ points that pass through a line, and $q + 1$ lines meet at each point. The resulting field-plane hexagon is thus always of diameter six [29]. The 21 edges in Fig. 14 correspond to the linear concatenation of three PGC(2, 2)'s, and the codeword of FPH (8, 13) has 8 data bits and 13 parity check bits. Assuming that all the edges drawn in dark lines represent information bits and the other edges correspond to parity bits, it can be seen that for each information bit, there exists a distinct cycle of size six consisting of five parity bits. Thus, FPH (8, 13) represents a coding of distant six, and thereby it can correct two errors (or detect three errors) in the codeword. In general, an $\mathrm{FPH}(2, q)$ has $m = q^3$ information bits and $2q^2 + 2q + 1$ check bits, and it can correct two errors. The coding efficiency of $\mathrm{FPH}(2, q)$ is $2/q + 2/q^2$. In order to find out an information bit is erroneous, altogether $2q - 1$ bits are scanned to estimate two parity bits, and by comparing the estimted values with those of actual bits in the codeword, a single error can be detected and corrected.

In order to understand how projective-geometry codes can be utilized in double-error correction in a DRAM, it is necessary to show that any arbitrary information bit and its associated parity bits can be accessed in one memory cycle. In this paper, we discuss a feasible organization in which an information bit can be read in a single memory cycle, and also it can be corrected if it is erroneous. The entire m information bits in a DRAM will be organized into $(m^{1/3}+1)$ separate $\mathrm{PGC}(2, m^{1/3})$'s, where each projective geometry will be composed of $m^{1/3} \times m^{1/3}$ information bits and $m^{2/3} + m^{1/3} + 1$ parity bits. Each PGC will be organized into a separate subarray, and altogether $m^{1/3} + 1$ subarrays will be needed. In each subarray $m^{1/3}$ information bits will be accessed together, because they will be on a single word line, and the first parity bit p_1 will be generated and compared with the corresponding bit in the codeword. In order to detect whether a particular information bit in the selected word line is faulty, $m^{1/3}$ information bits are selected from the different subarrays, one bit each. These information bits will be identified from the bipartite graph (field-plane hexagon) described by the points and lines of $\mathrm{PG}(2, m^{1/3})$. A second parity bit q_1 will be computed from these $m^{1/3}$ bits and it will be compared with the corresponding bit in the codeword. If both p_1 and q_1 are erroneous, then the information bit is faulty, and will be automatically corrected. The set of $m^{1/3}$ information bits needed to compute q_1 are located on the different word lines in the different subarrays, and thereby $m^{1/3}$ partitions become mandatory in an m-bit (nonredundant only) memory. Each subarray should have a special decoder

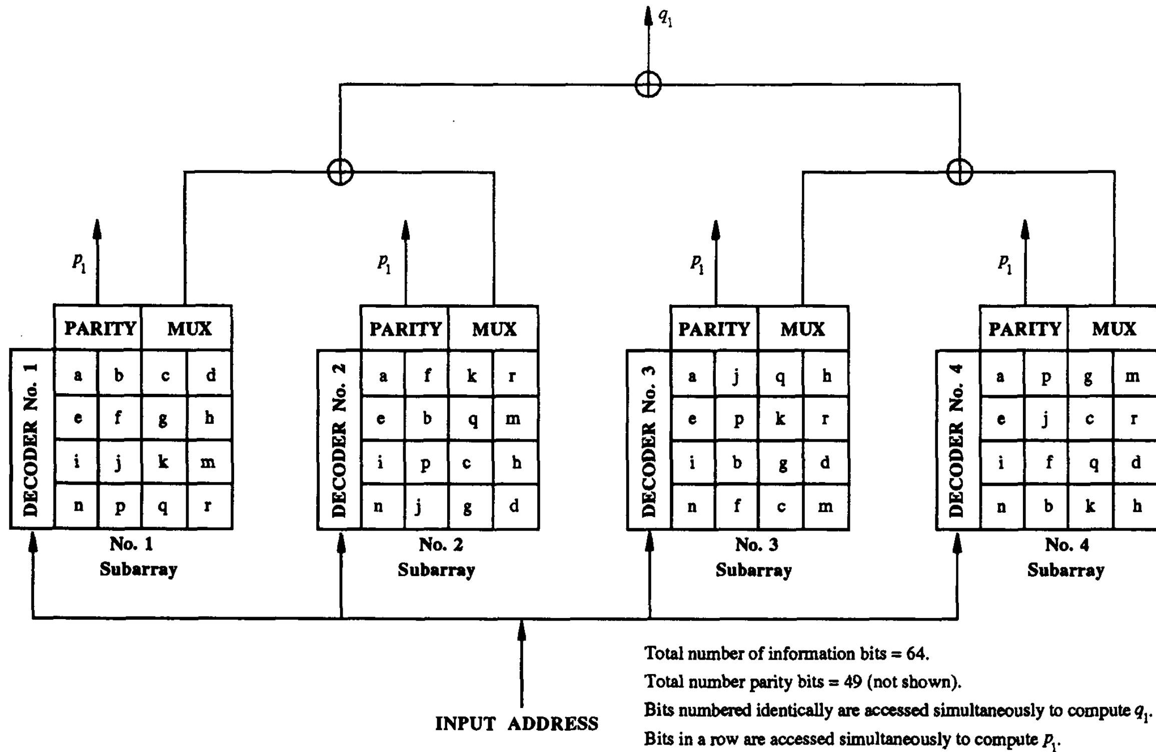


Fig. 15. Implementation of projective geometry code in DRAM.

that will compute over $\mathrm{GF}(m^{1/3})$ to identify these sets of $m^{1/3}$ bits. In Fig. 15, it is shown how the 64 information bits are organized into four subarrays, and how they are grouped into 16 sets such that cells numbered identically are selected in a single memory cycle for computing the parity bit. The main problem with this coding technique is that it divides an n-bit memory into $n^{1/3}$ subarrys, i.e., a 16-Mb DRAM will be organized into 256 subarrays. This is unrealistic because it will introduce very high decoder routing complexity, and also it will increase the chip area considerably. The practical 16-Mb DRAM manufactured by the NTT employs only eight partitions. Another problem with this design is that it can correct only two errors in the entire memory, and cannot correct common type faults such as a bit line stuck-at or short/open because of its defective sense amplifier. In Appendix B, a new type of code is proposed to circumvent the above limitations of the projective geometry code.

Appendix B

The $\mathrm{MTTF_{DEC}}$ is given by $\int_0^\infty R(t)\,dt = \int_0^\infty e^{-m^2\lambda t}[1 + m\lambda t + 0.5(m\lambda t)^2]^m$, where $\lambda = \phi\psi$. Let $y = m^2\lambda t$ so that $dy = m^2\lambda dt$. Thus,

$$\mathrm{MTTF_{DEC}} = \frac{1}{m^2\lambda}\int_0^\infty [1 + (y/m) + 0.5(y/m)^2]^m e^{-y}\,dy. \quad (1)$$

Let $y = m^{2/3}u$ so that $dy = m^{2/3}\,du$.

$$\mathrm{MTTF_{DEC}} = \frac{1}{m^2\lambda}\int_0^\infty ([1 + (u/m^{1/3}) + 0.5(u/m^{1/3})^2] \cdot e^{-u/m^{1/3}})^m m^{2/3}\,du. \quad (2)$$

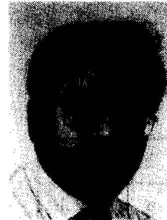
$$\mathrm{MTTF_{DEC}} \approx \frac{1}{m^2\lambda}\int_0^\infty \left(\left[\frac{1}{1 + u^3/3!}\right]^m m^{2/3}\,du. \quad (3)$$

$$\mathrm{MTTF_{DEC}} = \frac{1}{m^{4/3}\lambda}(3!)^{1/3}\Gamma(4/3). \quad (4)$$

References

[1] D. F. Barbe, *Very Large Scale Integration: Fundamentals and Applications.* Berlin: Springer-Verlag, 1980.
[2] L. L. Lewyn and J. D. Meindl, "Physical limits of VLSI dRAM's, *IEEE J. Solid-State Circuits,* vol. SC-20, pp. 231–241, Feb. 1985.
[3] J. Yamada, "Selector-line merged built-in ECC technique for DRAM's," *IEEE J. Solid-State Circuits,* vol. SC-22, pp. 868–873, Oct. 1987.
[4] A. H. Shah *et al.*, "A 4-Mbit DRAM with trench-transistor cell," *IEEE J. Solid-State Circuits,* vol. SC-21, pp. 618–627, Oct. 1986.
[5] T. Kaga *et al.*, "A 4.2μm ^{2} half-V_{CC} sheath-plate capacitor DRAM cell with self-aligned buried plate-wiring," *IEDM Dig.,* Oct. 1987, pp. 332–335.
[6] F. Horiguchi *et al.*, "Process technologies for high-density, high-speed 16M-bit dynamic RAM," in *Proc. Int. Conf. Electronic Devices Manufacturing,* Oct. 1987, pp. 324–327.
[7] P. Mazumder, J. H. Patel, and W. K. Fuchs, "Design and algorithms for parallel testing of random-access and content-addressable memories," *Design Automation Conf.,* vol. 24, pp. 688–694, July 1987.
[8] Y. You and J. P. Hayes, "A self-testing dynamic RAM chip," *IEEE J. Solid-State Circuits,* vol. SC-20, pp. 428–435, Feb. 1985.
[9] T. Sridhar, "A new parallel test approach for large memories," in *Proc. Int. Test Conf.,* 1985, pp. 462–470.
[10] A. Tuszynski, "Memory chip test economics," in *Proc. Int. Test Conf.,* 1986, pp. 190–194.

[11] P. Mazumder and J. H. Patel, "Parallel algorithms for parametric faults in DRAM," presented at 5th MIT Conf. Advanced Research in VLSI, Mar. 1988.
[12] J. S. Chern, P. Yang, P. Patnaik, and J. A. Seitchik, "Alpha-particle-induced charge transfer between closely spaced memory cells," *IEEE Trans. Electron Devices,* vol. ED-33, pp. 822–834, June 1986.
[13] R. J. McPartland, "Circuit simulations of alpha-particle-induced soft errors in MOS dynamic RAM's," *IEEE J. Solid-State Circuits,* vol. SC-16, pp. 31–34, Feb. 1981.
[14] T. C. May and M. H. Woods, "Alpha-particle-induced soft errors in dynamic memories," *IEEE Trans. Electron Devices,* vol. ED-26, pp. 2–9, Jan. 1979.
[15] G. A. Sai-Halasz, M. R. Wordeman, and R. H. Dennard, "Alpha-particle-induced soft-error rate in VLSI circuits," *IEEE J. Solid-State Circuits,* vol. SC-17, pp. 355–362, Apr. 1982.
[16] Meieren *et al.,* "Measurements of alpha particle radioactivity in IC device packages," in *Proc. 1979 Int. Reliabil. Phys. Symp.,* 1979, pp. 13–21.
[17] D. C. Yaney *et al.,* "Alpha-particle-tracks in silicon and their effects on dynamic MOS RAM reliability," *IEEE Trans. Electron Devices,* vol. ED-26, pp. 853–860, June 1979.
[18] T. Kubota *et al.,* "A new soft-error immune DRAM cell with a transistor on a lateral epitaxial silicon layer (TOLE cell)," in *IEDM Dig.,* Oct. 1987, pp. 344–347.
[19] T. Toyabe *et al.,* "A soft-error rate model for MOS dynamic RAM's," *IEEE J. Solid-State Circuits,* vol. SC-17, pp. 362–367, Apr. 1982.
[20] K. Shimogashi *et al.,* "An n-well CMOS dynamic RAM," *IEEE J. Solid-State Circuits,* vol. SC-17, pp. 344–348, Apr. 1982.
[21] F. I. Osman, "Error-correction techniques for random-access memories," *IEEE J. Solid-State Circuits,* vol. SC-17, pp. 877–881, Oct. 1982.
[22] P. Mazumder and J. H. Patel, "A novel fault-tolerant design of testable dynamic random-access memory," in *Proc. Int. Conf. Computer Design,* Oct. 1987.
[23] W. W. Peterson and E. J. Weldon, in *Error Correcting Codes.* Cambridge, MA: MIT Press, 1972.
[24] S. Lin, *An Introduction to Error-Correcting Codes.* Englewood Cliffs, NJ: Prentice-Hall, 1980.
[25] R. M. Tanner, "A recursive approach to low-complexity codes," *IEEE Trans. Inform. Theory,* vol. IT-27, pp. 533–547, Sept. 1981.
[26] J. Yamada *et al.,* "A 4-Mbit DRAM with 16-bit concurrent ECC," *IEEE J. Solid-State Circuits,* vol. 23, pp. 20–26, Feb. 1988.
[27] R. Carmichael, *Introduction to the Theory of Groups of Finite Order.* New York: Dover, 1956.
[28] R. M. Tanner, "A recursive approach to low-complexity codes," *IEEE Trans. Inform. Theory,* vol. IT-27, pp. 533–547, Sept. 1981.
[29] P. Dembwski, *Finite Geometries.* Berlin: Springer, 1968.



Pinaki Mazumder (S'84–M'87) received the B.S.E.E. from the Indian Institute of Science in 1976, the M.Sc. in computer science from the University of Alberta, Canada in 1985, and the Ph.D. in electrical and computer engineering from the University of Illinois at Urbana-Champaign in 1987.

Presently, he is working as an Associate Professor at the Department of Electrical Engineering and Computer Science of the University of Michigan at Ann Arbor. Prior to this, he worked two years as a research assistant at the Coördinated Science Laboratory, University of Illinois, and over six years in India at Bharat Electronics Ltd. (a collaborator of RCA) where he developed several types of analog and digital integrated circuits for consumer electronics products. During the summers of 1985 and 1986, he worked as a member of technical staff in the Naperville branch of AT&T Bell Laboratories. He is a recipient of Digital's Incentives for Excellence Award. He has published extensively on various aspects of VLSI research including testing, layout optimization ultrafast circuit design, and neural hardware. He is a Guest Editor of the IEEE Design and Test Magazine's special issue on multimegabit memory testing, published in June 1993. He is also editing Journal of Electronic Testing—Theory and Application's (JETTA's) special issue on advanced memory architectures and testing, which will appear in April 1994.

Dr. Mazumder is a member of IEEE, Sigma Xi, Phi Kappa Phi and ACM SIGDA.